US-PAT-NO: 6282602

DOCUMENT-IDENTIFIER: US 6282602 B1

TITLE: Method and apparatus for manipulating logical objects in
a data storage system

DATE-ISSUED: August 28, 2001

US-CL-CURRENT: 711/4, 711/111

APPL-NO: 09/ 107613

DATE FILED: June 30, 1998

---------- KWIC ---------

Primary Examiner - XP (1):
Lane; Jack A.

Brief Summary Text - BSTX (16):
FIGS. 3 and 4 help to describe in more detail the operations required to
copy the contents of file A to file B using a conventional computer system 100
having only a single mapping layer 220, for example, file system 222. As shown
in FIG. 3A, a typical file system manages large blocks of data including user

data 310, metadata 320, and free space 330. In FIG. 3A, the user data 310 represents that area of memory where user data corresponding to files is stored, and the free space 330 represents blocks of user data that are currently unused. The user data 310 and the free space 330 are shown as a set of contiguous logical blocks of memory 0 to 10,000 that are accessible by the file system. The term "logical block" is used to denote that blocks 0 to 10,000 may map directly or indirectly to blocks in physical space 230, depending on the number of levels of mapping between the file system and physical space. For a computer system having only two disks 241, 242 in the storage system 140, logical blocks 0 to 5,000 may correspond to physical blocks 0 to 5,000 on disk 241, while logical blocks 5,001 to 10,000 may correspond to physical blocks 0 to 5,000 on disk 242, although other mappings are possible. Those logical blocks of memory that are currently unused are circled in free space 330. Typically a table indicating blocks of free space 330 is stored (or cached) in the memory 130 of the host computer 110 (FIG. 1) so that blocks of free space 330 can be allocated quickly by the file system 222 operating on the host computer 110.

Brief Summary Text - BSTX (17):

The metadata 320 is also typically stored (or cached) in memory 130 of the host computer. The metadata 320 is used by the file system to keep track of the logical assignment of each block of user data 310. Typically, there is a metadata entry for each logical object owned by the file system. As shown in FIG. 3B, each metadata entry includes a number of fields of information, such as the name of the file, the date the file was created, the size of the file (e.g., in bytes), the location of the logical object at the next lowest layer, the level of protection assigned to the file, etc. In a computer system where there is no LVM and the file system 222 maps directly from application space 210 into physical space 230, the metadata entry provides the location and size, in physical space, of the named logical object (e.g., a file). In the example shown in FIG. 3B, the metadata entry tells the file system 222 that file A contains 514 bytes and that the file is stored on disk D1 (i.e., disk 241) at blocks 1 and 3 (where each block equals 512 bytes).

Brief Summary Text - BSTX (18):

FIG. 4 is a flowchart illustrating steps that are typically performed when the contents of a file A are copied to file B in a conventional computer system that includes a single mapping layer such as file system 222. A copy routine

can be called, for example, when an application issues the command to copy file A to file B. The copy routine proceeds to step 410 wherein the copy routine issues a file open command using the logical identifier file A. The file system looks to the metadata to determine if file A actually exists. If a metadata entry for file A does not already exist, the file open may report an error. Alternatively, if an entry for file A exists in the metadata, the file system returns a file handle for file A, i.e., a unique context to identify the routine's session of opening file A. The file handle or context of file A allows the file system to more quickly access the metadata of file A, and is often stored in a cache in the memory 130 of the host computer. Upon the successful return of a file handle for file A, the copy routine proceeds to step 420, where it issues a command to create file B. The file system again looks to the metadata to determine if file B already exists. If a metadata entry for file B does exist such that the writing of the contents of file A to file B would overwrite the data in file B, the routine may return an error or ask the user if the file should be overwritten. If a metadata entry for file B does not exist, the file system creates a metadata entry for file B, along with a file handle or context for the routine's session of opening file B. The metadata entry for file B will contain certain fields of information

**for file
B, such as its name, it creation date, etc. However, other fields of information in the metadata entry for file B will be empty (for example the
size of file B, the location of file B, etc.).**

Brief Summary Text - BSTX (19):

**After creating a metadata entry and file handle for file B, the copy command
proceeds to step 430, where it issues a read of file A. The read command
results in the file system determining the location in physical space of the
first portion of data in file A, which is determined in this example by
accessing the metadata entry for file A. It should be appreciated that if
additional mapping layers (e.g., LVM 224) were employed, the actual physical
location of the data would be determined by using the information provided by
the file system metadata as an index into mapping information (metadata or an
equivalent data structure) for the next lowest mapping layer, and that the
process would repeat until reaching the lowest mapping layer. In a
conventional computer system, the first portion of file A will typically
include one or more logically contiguous blocks of data up to some maximum
number of contiguous blocks. The maximum number of contiguous blocks and the
size of those blocks (e.g., in bytes) may vary based upon a number of factors**

including the operating system of the host computer, the file system, the storage system, and the interface by which the computer system is connected to the storage system. To simplify the example used herein, only one block is accessed at a time.

Brief Summary Text - BSTX (20):

After determining the real location of the first portion of data in file A, the first portion of data in file A (located on disk D1, block 1 in the example of FIG. 3B) is read from the appropriate physical device and returned to the copy routine, where it may be temporarily stored in memory 130 of the computer system. Next, the copy routine proceeds to step 440, where it issues a write of the data read in step 430 to file B. To perform the write, the file system accesses the metadata entry of file B via its context. Finding that the size and location fields are empty, the file system will proceed to get one or more blocks from free space 330 (e.g., logical block 0) to store this data. As logical block 0 corresponds to block 0 on disk D1 in the example shown, the data (from file A) is read from the memory 130 in the host computer and written to file B at block 0 of disk D1 in physical space 230. After writing the first portion of file B, the file system updates the metadata fields for file B. For

example, the file system will update the size of file B to reflect that it has 512 bytes of data (as only one block was written thus far), and that this first block is located on disk D1, block 0.

Brief Summary Text - BSTX (21):

The copy routine next proceeds to step 450, where the routine issues a read to get the next portion of user data for file A. As the file system is aware of what has previously been read from file A, the read command results in the file system determining the physical location of the next portion of data in file A by accessing the metadata field for file A in the manner described above. The remaining portion of file A is read and returned to the copy routine, where it may be temporarily written to the memory 130 of the host computer. The routine then proceeds to step 460, wherein the routine issues a write of the returned data to a physical location corresponding to file B. The file system accesses the metadata entry of file B, and proceeds to get another block from free space. In the example shown, the file system allocates the next block of available free space, block 2, for this purpose, although other allocation schemes may be used. After writing the remaining portion of data from the host computer to file B at disk D1, block 2, the file system updates the metadata

entry for file B. In this example, the metadata of file B is updated to indicate a size of 514 bytes, and a logical location of disk D1, blocks 0 and 2. After writing the remaining data to file B at step 460, the copy routine issues a file close command, at step 470, to close file A and file B, whereupon the copy routine terminates. When the copy routine closes file B, the updated metadata entry for file B is written back to its appropriate location on disk.

Detailed Description Text - DETX (7):

An application initiates the high speed copy routine by issuing a command to high speed copy file A to file B (e.g., Hcopy file A file B). At step 510, the Hcopy routine issues a file open command using the logical object identifier file A. As described previously with respect to FIG. 4, the file system may look for a metadata entry for file A to determine if file A actually exists for any of the mapping layers on the system (e.g., file system 222 or LVM 224 of FIG. 2). If a metadata entry for file A does not already exist in the metadata, the routine may report an error. When a metadata entry for file A exists, the file system can return a file handle for file A. However, it should be appreciated that the present invention is not limited in this respect, and can use any technique for accessing the metadata and identifying the logical

objects mapped therein.

Detailed Description Text - DETX (8):

Upon the successful return of a file handle for file A, the Hcopy routine proceeds to step 520, where it issues a command to create file B. As in a conventional copy routine, the file system may again look to the metadata to determine whether a metadata entry for file B already exists. If a metadata entry for file B exists such that the writing of the contents of file A to file B would overwrite the data in file B, the Hcopy routine may return an error or query the user whether the file should be overwritten. If a metadata entry for file B does not exist, then the file system creates a metadata entry for file B in the appropriate mapping layer (e.g., file system 222 in FIG. 2), and may create a file handle or context for file B. Once again, the metadata entry for file B may contain certain fields of information for file B, such as its name, its creation date, etc., although, other fields of information, such as the size and location of file B will be empty.

Detailed Description Text - DETX (9):

After creating a metadata entry for file B, the Hcopy routine proceeds to step 530, where the routine requests a mapping for file A by calling a mapping

routine such as the one shown in FIG. 6. As described below, the mapping routine of FIG. 6 returns the mapping of the logical object file A in what the host computer 110 perceives to be physical space, along with its size (e.g., in bytes). As discussed below, the storage system may also include one or more layers of mapping, such that the mapping layer 220 in the host computer 110 may not map all the way to the actual physical space. However, in one embodiment of the present invention, this mapping is not taken into account by the mapping routine, so that the location of the block or blocks of a logical object passed to the storage device are provided as outputs of the mapping layer that interfaces with the top mapping layer in the storage device. In the above-discussed example of FIG. 3B, the mapping routine will indicate that file A is physically stored at disk D1, blocks 1 and 3, and has a size of 514 bytes.

Detailed Description Text - DETX (10):

After receiving the mapping of the logical object file A, the Hcopy routine proceeds to step 540. Assuming that file B did not previously exist (i.e., file B is not being overwritten), at step 540 the routine requests the file system to extend or allocate it a number of blocks for file B sufficient to store the amount of data of file A. Some file systems are already capable of

pre-allocating a number of logical blocks to a logical object. Where a file system presently does not support such pre-allocation, the file system can be modified to provide this capability. When pre-allocating blocks to a particular logical object, the file system selects the requisite number of blocks from the available blocks in free-space 330 (FIG. 3A). Thereafter, the file system updates the metadata entry of file B. Using the previous example of FIGS. 3B and 4, the metadata entry for file B will indicate that file B is stored at disk DI, blocks 0 and 2, and has a size of 1024 bytes, i.e., the number of bytes in two logical blocks of data.

Detailed Description Text - DETX (12):

The icopy command can be implemented as an API that is issued by the host computer 110 and is supported by the storage system 140 to perform an internal copy within the storage device of one or more source physical blocks of data to one or more destination physical blocks of data. The icopy command may have the form: CMD [sequence_of_source_blocks, sequence_of_destination_blocks], where the sequence_of_source_blocks parameter is a list of source addresses uniquely identifying the storage device and location within the storage device where a block of source data is stored, and the sequence_of_destination_blocks

parameter is a list of destination addresses uniquely identifying the storage device and location within the storage device where the block of source data is to be written. However, it should be appreciated that the present invention is not limited to this or any other specific command format. In the above example, the host computer issues a command to icopy blocks 1 and 3 of disk D1 to blocks 0 and 2 of disk D1. In response to the icopy command, the storage system copies the data from disk D1, block 1 to disk D1, block 0, and the data from disk D1, block 3 to disk D1, block 2. After copying the physical blocks of data in step 560, in one embodiment of the invention, the storage device responds with the number of bytes actually written to the specified destination (i.e., 514 bytes). This information can be used (in step 560) to update the metadata entry for file B to indicate that file B has a size of 514 bytes, whereupon the routine proceeds to step 570. At step 570, the Hcopy routine issues a file close command to close file A and file B, whereupon the Hcopy routine terminates. After closing file B, the updated metadata for file B can be written back to its appropriate location, in a manner similar to that discussed previously with respect to FIG. 4.

Detailed Description Text - DETX (25):

After identifying the number of mapping layers that are associated with the logical object at step 620, the mapping routine proceeds to step 630, wherein the mapping routine determines, for the first mapping layer associated with the specified logical object, the mapping of the object to the next lowest layer in the mapping layer 220. For each mapping layer, this can be done, for example, by accessing the portion of the data structure for the mapping layer (e.g., file system 222 or LVM 224) that stores the metadata for the logical object (e.g., a file) passed to the mapping layer. There are a number of ways of determining where the metadata for a particular file is stored in the data structure of a file system or LVM. For example, the structure and location of the metadata can be obtained directly from the vendor of the mapping layer (e.g., file system 222 or LVM 224). Once the structure and location of the metadata for a mapping layer (e.g., a file system or an LVM) is known, the mapping routine can directly access the structure to access the information that provides it with a window into the next layer of mapping.

Detailed Description Text - DETX (65):

For example, for the file system mapping layer 1330, the information pertaining to the organizational structure of files and directories within the

file system 1335 can be determined by querying the operating system of the host computer (e.g., 110 of FIG. 1). Information identifying the mapping of logical objects of the file system to the next layer below can be determined by accessing the <u>metadata</u> for the file system as described previously with respect to FIG. 6.

US-PAT-NO: 6546458

DOCUMENT-IDENTIFIER: US 6546458 B2

TITLE: Method and apparatus for arbitrarily large capacity
removable media

DATE-ISSUED: April 8, 2003

US-CL-CURRENT: 711/114, 711/115

APPL-NO: 09/ 751572

DATE FILED: December 29, 2000

PARENT-CASE:

CROSS REFERENCE TO RELATED APPLICATIONS

The present invention is related to an application entitled Apparatus and Method for Writing and Reading Data to and From a Virtual Volume of Redundant Storage Devices, Ser. No. 09/638,205, filed Aug. 11, 2000, assigned to the same assignee, and incorporated herein by reference.

---------- KWIC ---------

Primary Examiner - XP (1):

Lane; Jack A.

Brief Summary Text - BSTX (2):

The present invention is directed to an apparatus and method for writing and
reading data to and from a virtual volume of redundant storage devices. In
particular, the present invention is directed to an apparatus and method in
which metadata is stored for every block in a superblock written to a plurality
of physical storage devices, such that a data volume may be easily rebuilt from
any arbitrary subset of the redundant storage devices.

Brief Summary Text - BSTX (10):

The present invention provides apparatus and method for writing and reading
data to and from a virtual volume of redundant storage devices. The apparatus
and method make use of metadata identifying the number of data storage devices
and number of redundancy storage devices in the virtual volume of redundant
storage devices. In addition, other metadata, such as the identity of the data
storage devices and parity storage devices may be utilized. The metadata is
stored with each block written to each of the storage devices. In the event of
a failure of a storage device, the metadata is modified to reflect the failure
and the storage device to which the data intended for the failed storage device

was written. In this way, if a failure of a storage device is encountered, each block in the virtual volume of redundant storage devices has enough information in the metadata to identify where to find the data that was intended for the failed storage device. Thus, reconstruction of data using redundancy information is not required.

Drawing Description Text - DRTX (16):

FIG. 13 is a table depicting the various types of metadata employed by this invention in order to allow the using systems to configure or re-configure arbitrarily large capacity removable media virtual volumes or virtual files or virtual linear address spaces.

Detailed Description Text - DETX (35):

There are several methods for synchronizing the discovery of a failed device with the writing of the data to a reduced set of drives using the second or fourth methods described above: 1) each block written for a superblock is self consistent and contains metadata that describes its relationship to all the other blocks in the superblock. Therefore, when a read is expecting to encounter a P2 block and instead encounters a block that is a data block (in FIG. 4A this would be data block 0), the RAIT system can, by convention or by

specifically changing the metadata or by adding change notation to the
metadata, assume that there has been an on the fly remapping of the use of the
devices. This remapping is reflected in the metadata that is stored in
subsequent superblocks; 2) at the point of failure, a new block is appended to
the end of each of the data and parity blocks already written in the superblock. This new block is only a metadata block. The inserted metadata
block describes the new mapping. An identical metadata block would then be
placed both before and after the block that was moved to an alternative drive.
When the subsystem reads the blocks from the various media at a later date, it
would encounter the inserted metadata description instead of the expected P2
block and from that, discover that there had been a remapping and use the
inserted block to understand the new structure and verify consistency. This
method is less desirable than the first method from a performance standpoint
since it requires additional writing of additional blocks. However it does
provide a greater degree of consistency checking. Both methods could be
supported in a single product with the choice being directed via installation
settings more dynamically done by policy statements communicated independently
to the subsystem at volume definition or even at mount time; and 3) another
method is to back-up each of the devices, reconstruct the

metadata in each block to reflect the new mapping, and write the data and parity information in the new mapping format. This approach is the least desirable since it requires significant delay for the rewrite.

Detailed Description Text - DETX (55):

In particular, the mechanism of the present invention provides a set of removable media maintained by a storage subsystem as a single logical entity for other systems accessing this subsystem. In the depicted example, tape is the depicted media, but any removable media or functionally removable devices containing media such as small disk drives may be used with the mechanism of the present invention. The set of removable media in these examples consists of n units for addressing performance requirements of data where n is greater than 0. The set also employs p units for addressing reliability requirements, where p is greater than or equal to 0. In these examples, customer data may be reformatted into collections of data also referred to as super blocks. A system of metadata is resident on each of the individual pieces or units of media. This metadata identifies the units that are members of a set and the relationship of each unit of data stored thereon (e.g., in a super block) to all other sub-sets of customer data in the set. Reassignment of

functions of each of the units of media may be achieved during the course of writing data to the media. This mechanism allows up to p units of media to be individually dropped from usage and individually returned to service in any order during the course of writing data to the media.

Detailed Description Text - DETX (57):

The mechanism of the present invention provides a transition from one set of removable media to another set of removable media. This transition may be gradual in which as little as one extra removable media drive is transitioned at a time. Alternatively, a setup of all of the media in the new set may be set up at one time. The value of n and p may be different for each set and changed during usage of the set of removable media. Additionally, in these examples, the system of metadata stored on an arbitrary media set k also identifies units of media in set k-1 and k+1. Set k-1 is the set of media prior to k while k+1 is the set of media after k. The terms \`prior\` and \`after\` can be determined by assessing the logical sequences of the data or by the temporal sequences of the data or by other algorithmic means. The system of metadata stored on the units of media in a particular set of media will also identify the following: (1) The number of sets that may be included

in the
definition of the single logical entity addressed by the system accessing the
sets of data or the boundaries established for such a number; (2) The number of
sets currently in use in the single logical entity addressed by the system
accessing the media and the placement within the boundaries; (3) The specific
units of media in each set in the definition, including individual values of n,
p, and other specific information to the set, such as unusable sections of
media; (4) The position of specific data in the sets. The number of sets
currently in use may be, for example, a setup to use only the first two sets
out of k possible sets, a setup to use the first and nth set of k possible
sets, or a setup to use the nth through the ith of k possible sets of removable
media. Unused sets do not require allocation of media units except as an
installation preference in these examples. The mechanism of the present
invention also allows an ability to move one item of data located in a specific
set of removable media to another item of customer data located in a different
set of removable media.

Detailed Description Text - DETX (63):
When the first unit of media in use gets to the first media staging point,
the use of the first unit of media, unit 1108, within set 1100 is

discontinued
and the first unit of media, unit 1122, in set 1102 is used. There are now two
concurrent sets of media in use and the metadata for all blocks must identify
the specific units of media mapped for use, both within the first set of media
units, set 1100, and within the second set 1102. The first unit of media, unit
1108, in set 1100 is unloaded from drive 1 within drive set 1164. The second
unit of media in set 1102 is loaded onto drive 1 which is now in drive set
1166. Immediately, the use of the second unit of media, unit 1110, in set 1100
is discontinued. The second unit of media, unit 1124 in set 1102 is used. The
metadata for all blocks now identify this unit of media as also being mapped
for use with the first unit of media in set 2 (media unit 1122) and the rest of
the set of media units in set 1100.

Detailed Description Text - DETX (66):

The process begins by determining whether metadata identifies the ability to
use more than one set of media units for a logical entity. If the metadata
does identify more than one set of media units can be used for the logical
entity, then a set of media units are set up to receive customer data and
identify the criteria for introduction of a subsequent set of media units (step
1202). This setup is made as appropriate for staging the

introduction of new media units. Data is accepted and written to the first set of media units (step 1204). A determination is then made as to whether the criteria identified that it is time to introduce media from the next set of media units (step 1206). If it is not time to introduce media from the next set of media units, the process returns to step 1204. Otherwise, a determination is made as to whether the introduction of media from the next set of media units is an immediate swap or a staged swap of media units (step 1208). If the introduction is to be a full immediate swap, then the next full set of media is brought up and customer data is buffered as needed to finish the setup of the next full set of media (step 1210) with the process then returning to step 1204. If the introduction of media from the next set of media units is a staged swap, then the use of one or more units is discontinued in the present set of media and one or more appropriate units from the next set of media is introduced for use with the current set (step 1214) with the process then returning to step 1204 as described above.

Detailed Description Text - DETX (67):

With reference again to step 1200, if the metadata does not identify more than one set of media units for the logical entity, then a standard

RAIT process is used to write customer data until the logical device is declared full or all customer data is written (step 1212) with the process terminating thereafter.

Detailed Description Text - DETX (68):

Turning next to FIG. 13, a diagram of types of metadata required for this system to be fully functional is depicted in accordance with a preferred embodiment of the present invention. The metadata is associated with the system in several different ways. One set of metadata is generally included with each super block of data written on any piece of media and identifies the relationship of the specific super blocks written as a contemporary set including media unit locations and functional use such as application data of redundancy data. This metadata can also be inferred with a flag that indicates that the metadata has not changed and can be algorithmically derived from the previous set of metadata. This will reduce the amount of overhead for metadata storage significantly. When there is a change in the algorithmic metamorphosing of the metadata (e.g., a device has stopped working and must be mapped out of the rotation), the flag indicates the presence of the changed metadata. Another set of metadata is generally only required in few strategic

locations on each unit of media. The location of this metadata is dependent on the architecture of the media unit. A longitudinal tape, which writes to the end and then will rewind might record this metadata at the load point and again at end of tape. A serpentine tape which writes down and then back so the end and the beginning are at the same point on the tape might only record the metadata once. Using systems will find greater utility if the subsystem records this metadata at other strategic locations like when file markers are written to tape. This metadata includes the items noted as once per piece of media in FIG. 13.

US-PAT-NO: 6026474

DOCUMENT-IDENTIFIER: US 6026474 A

TITLE: Shared client-side web caching using globally addressable memory

DATE-ISSUED: February 15, 2000

US-CL-CURRENT: 711/202, 707/10 , 707/104.1 , 709/201 , 709/203 , 711/121
, 711/147

APPL-NO: 08/ 848971

DATE FILED: May 2, 1997

PARENT-CASE:

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation-in-part of co-pending U.S. patent applications Ser. No. 08/754,481, filed Nov. 22, 1996, and Ser. No. 08/827,534, filed Mar. 28, 1997, now U.S. Pat. No. 5,918,229, both of which are incorporated herein by reference.

---------- KWIC ---------

Primary Examiner - XP (1):

Lane; Jack A.

Detailed Description Text - DETX (32):
Directory entry scanning is one of the most frequently performed operations by user applications. It is also may be the most visible operation in terms of performance. Consequently, much attention is directed to making the directory scan efficient and the WindowsNT Files System (NTFS) duplicates sufficient file Inode information in the directory entry such that a read directory operation can be satisfied by scanning and reading the directory entries without going out to read the information from the file Inodes. The problem with this scheme is that the doubly stored file metadata, such as the file time stamps and file size, can be updated quite frequently, making the metadata update more expensive. However, this overhead is considered acceptable in face of the performance gained in directory scan operations.

Detailed Description Text - DETX (37):
A file of the file system 60 comprises streams of data and the file system metadata to describe the file. Files are described in the file system 60 by objects called Inodes. The Inode is a data structure that stores the file metadata. It represents the file in the file system 60.

Detailed Description Text - DETX (38):

A data stream is a logically contiguous stream of bytes. It can be the data stored by applications or the internal information stored by the file system 60. The data streams are mapped onto pages allocated from the addressable shared memory space 20 for storage. The file system 60 segments a data stream into a sequence of 4 kilobyte segments, each segment corresponding to a page. The file system 60 maintains two pieces of size information per data stream: the number of bytes in the data stream, and the allocation size in number of pages. The byte-stream to segment/page mapping information is part of the file <u>metadata</u> and is stored in a structure called data stream descriptor. See FIG. 4.

Detailed Description Text - DETX (114):

As further depicted in FIG. 9, each directory page 120 includes a page header 322 that includes attribute information for that page header, which is typically <u>metadata</u> for the directory page, and further includes directory entries such as the depicted directory entries, 324 and 326, which provide an index into a portion of the shared address space wherein that portion can be one or more pages, including all the pages of the distributed shared memory

space. The depicted directory page 320 includes directory entries that index a
selected range of global addresses of the shared memory space. To this end,
the directory generator can include a range generator so that each directory
entry can include a range field 330 that describes the start of a range of
addresses that that entry locates.